INFORMATION AND COMPUTATION 89, 180–198 (1990)

Efficient Solution of Some Problems in Free Partially Commutative Monoids*

HAI-NING LIU

Department of Computer Science, University of California, San Diego, La Jolla, California 92093

AND

C. WRATHALL AND KENNETH ZEGER†

Department of Mathematics, University of California, Santa Barbara, California 93106

Linear-time algorithms are presented for several problems concerning words in a partially commutative monoid, including whether one word is a factor of another and whether two words are conjugate in the monoid.

1. INTRODUCTION

In theoretical computer science, and, in particular, in automata and formal language theory, questions arise concerning words and sets of words in free monoids. Results and techniques from the study of the combinatorial algebra of the free monoid have provided an "algebraic" basis for dealing with many of these questions. In recent years there has been increasing interest in properties of words and sets of words in free partially commutative monoids or commutation monoids, that is, monoids in which some pairs of generating letters commute but no other relations hold. Some of this interest is due to the use of free partially commutative monoids in modeling problems of concurrency control in databases and parallel computation, as, for example, by Flé and Roucairol (1985); in a recent survey, Aalbersberg and Rozenberg (1988) have presented the connections between free partially commutative monoids and Petri nets. As in the case of free monoids, problems about words are linked to combinatorial properties, and the combinatorial algebra of free partially commutative monoids has

* The research reported here was supported in part by the National Science Foundation under Grants DCR-8314977, CCR-8611980, and CCR-8706976.

† Current address: Department of Electrical Engineering, University of Hawaii, Honolulu, Hawaii 96822.

0890-5401/90 $3.00

received much attention since arising in the work of Cartier and Foata (1969) on rearrangements.

Formally, a free partially commutative monoid is presented by a (finite) alphabet and a binary relation on the alphabet that specifies which letters are "independent" or commute. The monoid itself is then the quotient of the free monoid on the given alphabet by a congruence relation on strings derived from the independence relation on letters. For any such structure, and especially one to be used as a model, the issue naturally arises of the algorithmic solution and inherent complexity of questions about the monoid: such questions as whether two words in the underlying free monoid represent the same element of the monoid (the Word Problem), and whether two presentations give rise to isomorphic monoids.

For free partially commutative monoids, dealing with algorithmic questions is aided by the fact that elements of the monoid ("traces") can be translated (faithfully) from representative words to certain tuples of words (Cori and Perrin, 1985; Perrin, 1986; Duboc, 1986b) or to certain directed acyclic graphs (Cori and Metivier, 1985). Using one notion of projection to a tuple of words, for example, Book and Liu (1987) have given a linear-time (Turing machine) algorithm for the Word Problem for free partially commutative monoids. The algorithms here also use that notion of projection, in the form given explicitly by Cori and Perrin (1985); a similar notion was used by Keller (1973). A different scheme was used by Perrin (1986) to produce a normal form from a given word, which also gives rise to a linear-time algorithm for the Word Problem.

Much recent work on free partially commutative monoids has treated their rational and recognizable subsets (which need not be the same class). One algorithmic question in this context is whether a given word is congruent to any word in some regular set; Bertoni *et al.* (1982) have described a polynomial-time algorithm to answer this question and have shown that when the input includes specifications of the monoid and regular set as well, the question is NP-complete.

The Isomorphism Problem—whether two free partially commutative monoids presented as such are isomorphic—is solvable, but is not known to be tractable, since it is equivalent to the Isomorphism Problem for (finite) undirected graphs. At the extreme cases of free abelian monoids and free monoids, the graphs take the simple forms of complete graphs and graphs with no edges, respectively, and so the problem is easily solved in those cases.

In this paper, we present linear-time algorithms for certain problems concerning either single words or pairs of words in a fixed free partially commutative monoid. The time bounds for the algorithms are linear functions of the length of the input words, under the RAM model of computation (although the Turing machine model will sometimes suffice).

The purpose is to make explicit the existence of efficient solutions for the problems, and the algorithms rely on the algebraic properties of free partially commutative monoids developed by others and on algorithms for words in free monoids (Knuth *et al.*, 1977; Avenhaus and Madlener, 1980). The principal results concern finding the root of a given word, testing whether one word is a factor of another, and testing whether two words are conjugate.

There is a natural extension of the notion of "primitive root" of a word from free monoids to free partially commutative monoids, and that "congruential root" can be found in linear time (Theorem 4.2). As a consequence, it can be tested in linear time whether two words have any common powers (Corollary 4.3).

For a pair of words, it can be asked whether one represents a factor of the other in the free partially commutative monoid, that is, whether there is some allowable rearrangement of the letters in the second word for which the first word appears as a block. This question can be answered, and an appropriate factorization (if any) found, in linear time (Theorem 5.3). The solution to this "pattern-matching" problem generalizes the linear-time solution in free monoids using the Knuth–Morris–Pratt algorithm, and is based on it.

There are several ways to extend the notion of conjugacy from free monoids to arbitrary monoids and the two that seem most natural for free partially commutative monoids are considered here: "conjugacy" and "cyclic equality." (The first is an equivalence relation but the second might not be.) Whether two words stand in either of these relations can be tested in linear time and, when they do, appropriate witnesses are found (Theorem 6.3). For these algorithms, the problems are recast as questions of factorization in the free partially commutative monoid.

Because the size of the alphabet (or, more precisely, the number of commuting pairs of letters) contributes to the "constant" factor in the linear time bounds, the corresponding uniform problems, in which a specification of the monoid is part of the input, may not have linear-time solutions but are solvable in polynomial time. For example, the uniform factorization problem can be solved in time bounded by the square of the length of the input.

2. Preliminary Definitions and Notation

For a set A, $|A|$ denotes the cardinality of A and $\bar{A}$ denotes the complement of A (with respect to the appropriate universe).

For an alphabet (set of letters) Σ, Σ^* denotes the free monoid generated by Σ, with the empty word denoted by e. We are concerned here only with

finite alphabets. A development of the basic properties of words in a free monoid can be found in Chapter 1 of the book of M. Lothaire (1983).

For each letter $a \in \Sigma$ and word $w \in \Sigma^*$, $|w|_a$ denotes the number of occurrences of a in w. The *length* of word w, $|w|$, is the total number of occurrences of letters in w, so that $|w| = \Sigma_a |w|_a$. The set of letters occurring in w is denoted by alph(w).

For words x, y, and z, if $x = yz$, then y is a *prefix* of x, and z is a *suffix* of x; if $x = uyv$ for some words u and v, then y is a *factor* of x. The relation "y is a prefix of x" is also written as "$y \leqslant x$." If $x = uv = v'u$ with $v \neq e$ then u is an *overlap* of x (that is, a proper prefix and suffix of x).

A nonempty word x is *primitive* if there is no word y and integer $k > 1$ such that $x = y^k$. The *root* $\rho(x)$ of $x \neq e$ is the (unique) primitive word r such that $x = r^k$ for some $k \geqslant 1$.

Let $\mathbf{N}$ denote the natural numbers $\{0, 1, 2, \ldots\}$. For an alphabet Σ listed as $\{a_1, \ldots, a_m\}$, the corresponding *Parikh mapping* is the function $\Psi: \Sigma^* \to \mathbf{N}^m$ such that $\Psi(x) = (|x|_{a_1}, \ldots, |x|_{a_m})$. The Parikh mapping is a monoid homomorphism: $\Psi(xy) = \Psi(x) + \Psi(y)$, where addition is extended componentwise to $\mathbf{N}^m$. Also extending the usual order on $\mathbf{N}$, write $\Psi(x) \leqslant \Psi(y)$ if and only if, for all $a \in \Sigma$, $|x|_a \leqslant |y|_a$.

Several of the algorithms presented here are based on efficient string-matching algorithms for free monoids that use the concept of "failure functions" of words. The failure function of a word $a_1 \cdots a_n$, $a_i \in \Sigma$, is the function $f: \{1, \ldots, n\} \to \{0, \ldots, n-1\}$ for which $f(k)$ is the largest $j < k$ such that $a_1 \cdots a_j$ is a suffix of $a_1 \cdots a_k$. (In other words, $f(k)$ is the length of the longest overlap of $a_1 \cdots a_k$.) The failure function of x can be computed in time linear in $|x|$, and it can be used to find all occurrences of x in another word y in time linear in $|xy|$. Details can be found in the text by Aho, Hopcroft, and Ullman (1974), as can a description of the RAM and Turing machine models of computation.

The failure function of a word can also be used to find its root in linear time, due to the fact expressed in the following proposition. Essentially the same observation has been made by Avenhaus and Madlener (1980) in presenting algorithms for root, conjugacy, and other problems in free groups; however, the proof is presented here for completeness.

PROPOSITION 2.1. *Suppose u is the longest overlap of a nonempty word x and $x = uv = v'u$. If v is a prefix of u then $v = \rho(x)$; otherwise, x is primitive.*

Proof. Let $r = \rho(x)$ and suppose $x = r^{m+1}$, $m \geqslant 0$. Since r^m is an overlap of x and u is the longest overlap of x, $|r^m| \leqslant |u| < |r^{m+1}|$.

If v is a prefix of u then (since $x = uv = v'u$ and $|v'| = |v|$), $v = v'$ and $x = uv = vu = r^{m+1}$ with r a primitive word; hence $v = r^j$ for some $j \geqslant 0$

(Lothaire, 1983, Props. 1.3.1, 1.3.2). Since $|r^m| \leqslant |u| < |r^{m+1}|$, $j = 1$ and so $v = r$.

If $m \geqslant 1$ then (since $x = r^{m+1} = uv$ and $|r^m| \leqslant |u|$) there is some word s such that $u = r^m s$ and $r = sv$. Hence $x = (sv)^{m+1} = v'u = v'(sv)^m s = v's(vs)^m$, so, since m is positive, $r = sv = vs$ and then $u = r^m s = vsr^{m-1}s$ and thus v is a prefix of u. ∎

Pattern-matching problems for free monoids can also be approached from other directions. Galil and Seiferas (1983) have presented an algorithm that tests whether one word is a factor of another and that uses linear time on a multihead finite automaton.

3. Projection and Reconstruction

This section gives the definition of free partially commutative monoids and the basic facts concerning their representation by projections. Each word x over the alphabet generating the free partially commutative monoid has an associated family of projections, a tuple of words $\Pi(x)$, and two words give rise to the same family exactly when they represent the same element of the monoid. For an arbitrary family, it can be asked whether it arises from some word (i.e., is "reconstructible"); this question can be answered in linear time and, if the answer is "yes," any or all such words can be found (Theorem 3.3).

Definition. A *partially commutative alphabet* is a pair (Σ, θ), where Σ is an alphabet and $\theta \subseteq \Sigma \times \Sigma$ is a symmetric and irreflexive relation (that is, $(a, b) \in \theta$ implies $(b, a) \in \theta$ and for no $a \in \Sigma$ is (a, a) in θ). The pairs of (distinct) letters that commute are given by θ, and the noncommuting pairs are given by $\bar{\theta} = (\Sigma \times \Sigma) - \theta$. For $a \in \Sigma$, let $\theta(a) = \{b \in \Sigma: (a, b) \in \theta\}$ and $\bar{\theta}(a) = \{b \in \Sigma: a \neq b, (a, b) \in \bar{\theta}\}$.

The pair $(\Sigma, \bar{\theta})$ can be considered as an undirected graph on vertices Σ with an edge between every pair of distinct letters that do not commute. The set $\bar{\theta}(a)$ consists of the neighbors (that is, adjacent vertices) of the letter a in that graph.

A partially commutative alphabet determines a monoid as follows.

Definition. Let $\leftrightarrow$ be the relation on Σ^* defined by $xaby \leftrightarrow xbay$ for all $x, y \in \Sigma^*$ and all $(a, b) \in \theta$, and let $\equiv$ be the reflexive, transitive closure of $\leftrightarrow$. The relation $\equiv$ is a congruence relation on Σ^*, "congruence modulo θ." The *free partially commutative monoid* determined by (Σ, θ) is the quotient monoid $\Sigma^*/\equiv$ and is denoted by $M(\Sigma, \theta)$.

The elements of the monoid $M(\Sigma, \theta)$ are the congruence classes $[u]$ for $u \in \Sigma^*$, where $[u] = \{v \in \Sigma^* : v \equiv u\}$; the class $[e]$ serves as the identity element of the monoid and the product is given by $[u] \cdot [v] = [uv]$. Two words are congruent exactly when one can be obtained from the other by a sequence of exchanges of adjacent commuting letters. In particular, congruent words have the same Parikh mapping.

The projection of a word with respect to a pair of letters is the word formed by erasing all letters in the word except for the chosen pair. Of particular interest are projections of words with respect to pairs of noncommuting letters.

DEFINITION. For $a, b \in \Sigma$ (not necessarily distinct), $\pi_{ab} : \Sigma^* \to \{a, b\}^*$ is the homomorphism determined by defining $\pi_{ab}(a) = a$, $\pi_{ab}(b) = b$, and $\pi_{ab}(c) = e$ for $c \neq a, b$. Let $\Pi(x)$ denote the family $(\pi_{ab}(x) : (a, b) \in \bar{\theta})$.

Since θ is an irreflexive relation, the projection $\pi_{aa}(x)$, counting the number of occurrences of a in x, is included in the family $\Pi(x)$. For $a \in \Sigma$ and $b \in \bar{\theta}(a)$, both (a, b) and (b, a) are included in the index set of $\Pi(x)$ even though just one would be sufficient (since $\pi_{ab}(x) = \pi_{ba}(x)$); the redundancy is for notational convenience only.

The following characterization of congruence modulo θ in terms of projections provides the basis for many of the following results and algorithms. One easily proved consequence is that $M(\Sigma, \theta)$ is a cancellative monoid (that is, $uxv \equiv uyv$ implies $x \equiv y$.)

PROPOSITION 3.1. (Cori and Perrin, 1985, Prop. 1.1). *For all words u and v, $u \equiv v$ if and only if $\Pi(u) = \Pi(v)$.*

Families of words indexed by $\bar{\theta}$ arise that are not necessarily formed by projecting a word. Such a family $U = (U_{ab} : (a, b) \in \bar{\theta})$ is assumed to be symmetric (i.e., $U_{ab} = U_{ba}$) and to have components on the correct alphabets (i.e., $U_{ab} \in \{a, b\}^*$). Concatenation is extended to families componentwise, so that, for $V = (V_{ab} : (a, b) \in \bar{\theta})$, $U \cdot V = (U_{ab} V_{ab} : (a, b) \in \bar{\theta})$; note that $\Pi(xy) = \Pi(x) \cdot \Pi(y)$. The "length" of the family U is the sum of the lengths of the components of U.

DEFINITION. A family $Y = (Y_{ab} : (a, b) \in \bar{\theta})$ is *reconstructible* if there is some word x such that $Y = \Pi(x)$. The family Y is *quasi-reconstructible* if for every $a \in \Sigma$ and every $b \in \bar{\theta}(a)$, $|Y_{ab}|_a = |Y_{aa}|$.

Establishing that families are reconstructible is often made easier by use of the following facts due to Duboc (1986b, Props. 1.6, 1.7) and Cori and Metivier (1985, Prop. 3.8).

PROPOSITION 3.2. (a) *If $Y \cdot Z$ is reconstructible and Y (or Z) is quasi-reconstructible then both Y and Z are reconstructible.*

(b) *If Y^k is reconstructible for some $k \geqslant 1$ then Y is reconstructible.*

A multitape Turing machine can check a given family for reconstructibility and, if it is reconstructible, produce a "reconstructed" word; the time taken need be no greater than linear in the total length of the family.

Reconstruction Procedure. On input $Y = (Y_{ab}: (a, b) \in \tilde{\theta})$:

1. Check that Y is quasi-reconstructible (and has the correct form).
2. If all components of Y are empty, halt and accept.
3. Form

$$F = F(Y) = \{a \in \Sigma: \text{for all } b \text{ such that } (a, b) \in \tilde{\theta},\ a \leqslant Y_{ab}\}.$$

If F is empty, halt and reject. Otherwise, let $a \in \Sigma$ be any element of F. Print a, remove the first letter from each entry Y_{ab} and Y_{ba} for $b \in \tilde{\theta}(a) \cup \{a\}$, and return to (2).

THEOREM 3.3. *On input Y, the Reconstruction Procedure operates in time linear in the length of Y. Input Y is rejected if Y is not reconstructible; if Y is reconstructible, the output printed is a word y such that $Y = \Pi(y)$.*

Proof. It can be assumed that the family is presented with each component on a separate tape. Checking, for all appropriate letters a and b, whether $Y_{ab} \in \{a, b\}^*$, $Y_{ab} = Y_{ba}$, $|Y_{ab}|_a = |Y_{aa}|$, and $|Y_{ab}|_b = |Y_{bb}|$, can be done by simply reading the tapes. The set $F(Y)$ is apparent from the first letters of the components, so Step 3 requires constant time (since Σ and θ are fixed). The overall linear time bound then follows since the length of at least one component is reduced each time Step 3 is performed.

In Step 3, the first letter of Y_{ab} and Y_{ba} must be a, since a belongs to F. Beginning with a quasi-reconstructible family Y_0, if x has been printed and the remaining family is Y_1, then $Y_0 = \Pi(x) \cdot Y_1$ and Y_1 is quasi-reconstructible; in particular, Y_0 is reconstructible if and only if Y_1 is. Therefore, if the procedure, on input Y, halts and accepts after printing word y then $Y = \Pi(y)$. On the other hand, if the procedure halts and rejects then the original input was not reconstructible, since, for a reconstructible family $Z = \Pi(z)$, $F(Z)$ is the set $\{a \in \Sigma: z \equiv az' \text{ for some } z'\}$ and so is non-empty. ∎

Suppose the alphabet Σ has a total order that is extended lexicographically to Σ^*. If the least element of F is always chosen in Step 3, then the procedure, on input $\Pi(x)$, will print the least word congruent to x. If, instead, the entire set F is printed as a block (or in order), the procedure

will print the Foata Normal Form of x (Lallement, 1979, Chap. 11; Perrin, 1986).

Another variation on this procedure can be used for a multitape Turing machine that, given $\Pi(x)$, calculates and prints the congruence class of x in time linear in the length of its output (that is, linear in $|x| \cdot |[x]|$). It is only necessary to add a stack to hold the remaining possibilities for letters to be printed at each step; after completing the printing of each element of $[x]$, the machine can back up to begin printing the next element. If the letters are dealt with in a fixed order, the congruence class will be printed in the corresponding lexicographic order. This process can also be viewed as producing all possible topological sorts of a certain directed acyclic graph derived from x (Cori and Metivier, 1985, Prop. 2.1).

The family of sets $\{\{a, b\}: (a, b) \in \bar{\theta}\}$ consists of one- and two-element cliques in the graph $(\Sigma, \bar{\theta})$; families of larger cliques (that cover the graph) can be used to characterize the congruence relation in a manner similar to that given in Proposition 3.1 (Duboc, 1986b, Prop. 1.2). The results here could be stated in that more general framework, although the small cliques seem most convenient for the pattern-matching algorithm in Section 5.

4. Roots and Powers

For an element $[x]$, $x \neq e$, in a free partially commutative monoid, there is a unique "primitive" element $[r]$ such that x is congruent to a power of r; words in $[r]$ are "congruential roots" of x. The notions of "primitive" and "root" are analogous to those in free monoids. This section presents a linear-time algorithm to find the congruential root of a given word, the basis of which is determining the primitive roots (in free monoids) of the projections of the given word. A related question is whether two words have any common powers, and it, too, can be solved in linear time.

Definition. A word $x \neq e$ is *imprimitive mod* θ if there exist $k > 1$ and $y \in \Sigma^*$ such that $x \equiv y^k$; otherwise x is *primitive mod* θ. If $x \equiv y^k$ and y is primitive mod θ then y is a *congruential root* of x.

Although this definition allows different words to be congruential roots of the same word, the following proposition shows that any two congruential roots of a word are themselves congruent. The second part of the proposition follows immediately from the first part and the definition.

Proposition 4.1. (Duboc, 1986a, Thm. 1.5; 1986b, Cor. 2.2). (a) *For nonempty words x and y, there exist $n, m \geqslant 1$ such that $x^n \equiv y^m$ if and only if there exist $j, k \geqslant 1$ and $z \in \Sigma^*$ such that $x \equiv z^j$ and $y \equiv z^k$.*

(b) *For $x \neq e$, if y_1 and y_2 are congruential roots of x then $y_1 \equiv y_2$.*

Notation. For $x \in \Sigma^*$, $\rho_\theta(x)$ denotes "the" congruential root of x, that is, any of the words in that congruence class, with $\rho_\theta(e) = e$. Recall that $\rho(x)$ denotes the root of x in the free monoid $\text{alph}(x)^*$.

THEOREM 4.2. *For a fixed partially commutative alphabet (Σ, θ), the congruential root of a word $x \in \Sigma^*$ can be found from x in time linear in $|x|$.*

Proof. For convenience, assume that $\text{alph}(x) = \Sigma$; otherwise, letters not occurring in x can be dropped from Σ and $\bar{\theta}$. With this assumption, each projection $\pi_{ab}(x)$ is nonempty.

Given x, form a collection $R = (R_{ab} : (a, b) \in \bar{\theta})$ as follows.

(1) For each pair $(a, b) \in \bar{\theta}$, find the root and the exponent of $\pi_{ab}(x)$ in the free monoid $\{a, b\}^*$, that is, the word $y_{ab} = \rho(\pi_{ab}(x))$ and the integer $n(ab) \geqslant 1$ such that $\pi_{ab}(x) = y_{ab}^{n(ab)}$.

(2) Find the greatest common divisor, d, of the integers $\{n(ab) : (a, b) \in \bar{\theta}\}$, and set $R_{ab} = y_{ab}^{k(ab)}$, where $k(ab) = n(ab)/d$.

As noted in Section 2, the root of a word w in a free monoid can be found in time linear in $|w|$ by use of the failure function of w. Once the root of each projection is known, a multitape Turing machine requires only time linear in $|x|$ to calculate the exponents (in unary), find their greatest common divisor, and form the entries in R.

As is justified by the following claim, the family R can be reconstructed into $\rho_\theta(x)$. Thus, the congruential root of a given word can be found in linear time.

CLAIM. $R = \Pi(\rho_\theta(x))$.

Proof. Since $R^d = \Pi(x)$, from Proposition 3.2(b) there is some v such that $R = \Pi(v)$ and $x \equiv v^d$. If v is imprimitive mod θ then there is some $z \in \Sigma^*$ and some $m \geqslant 2$ such that $v \equiv z^m$. This implies that, for all $(a, b) \in \bar{\theta}$, $y_{ab}^{k(ab)} = \pi_{ab}(v) = \pi_{ab}(z)^m$, so that, since y_{ab} is primitive in $\{a, b\}^*$ and nonempty, m divides $k(ab)$. However, by construction the integers $\{k(ab) : (a, b) \in \bar{\theta}\}$ are relatively prime, contradicting the assumption that $m \geqslant 2$. Thus, v is primitive mod θ, and $R = \Pi(v) = \Pi(\rho_\theta(x))$. ∎

The "Generaized Word Problem" is to determine whether two given words have a common power. Using Proposition 4.1(a), this problem reduces to checking whether the given words have the same congruential root.

COROLLARY 4.3. *The Generalized Word Problem "Given x and y, do there exist positive integers n and m such that x^n is congruent to y^m?" can be solved in time linear in $|xy|$.*

5. PATTERN-MATCHING

In a free monoid, pattern-matching problems ask whether a "pattern" word y occurs in a "text" word x, that is, whether y is a factor of x. For words on a partially commutative alphabet (Σ, θ), the pattern y might not appear directly in the text x, but instead in some word congruent to x; the question thus becomes whether $[y]$ is a factor of $[x]$ in the monoid $M(\Sigma, \theta)$. This question can be answered, and a factorization $x \equiv uyv$ (if any) found, in linear time, by making use of linear-time algorithms for pattern-matching in free monoids. The restricted question of whether the pattern is an "initial" factor of the text also has a linear-time solution.

DEFINITION. If there is some $z \in \Sigma^*$ such that $xz \equiv y$, then x is a *congruential prefix* of y, written $x \leqslant_\theta y$. If $y \equiv uxv$ for some u, v, then x is a *congruential factor* of y.

As a consequence of the first part of the following proposition, it can be easily tested whether one word is a congruential prefix of another. The relation "prefix-of" in free monoids (denoted by $\leqslant$) is extended componentwise to families of projections. Recall that the Parikh mapping Ψ counts the number of occurrences of each letter in a word.

PROPOSITION 5.1. *For all words x, y:*

(a) $y \leqslant_\theta x$ *if and only if* $\Pi(y) \leqslant \Pi(x)$;

(b) *if* $\Psi(y) \leqslant \Psi(x)$ *and* $yw \equiv xz$ *for some words* w, z *then* $y \leqslant_\theta x$.

Proof. (a) If $yz \equiv x$ for some z, then $\Pi(y) \cdot \Pi(z) = \Pi(yz) = \Pi(x)$, so every component of $\Pi(y)$ is a prefix of the corresponding component of $\Pi(x)$, and thus $\Pi(y) \leqslant \Pi(x)$. If, on the other hand, $\Pi(y) \leqslant \Pi(x)$ then for all $(a, b) \in \bar{\theta}$, there is some word z_{ab} such that $\pi_{ab}(y)\, z_{ab} = \pi_{ab}(x)$. For the family $Z = (z_{ab} : (a, b) \in \bar{\theta})$, we have $\Pi(y) \cdot Z = \Pi(x)$, so, from Proposition 3.2(a), there is some z such that $Z = \Pi(z)$ and hence $\Pi(yz) = \Pi(x)$ and $yz \equiv x$.

(b) From part (a), it is sufficient to show that $\Pi(y) \leqslant \Pi(x)$. For any $(a, b) \in \bar{\theta}$, $\pi_{ab}(y)\, \pi_{ab}(w) = \pi_{ab}(x)\, \pi_{ab}(z)$ and $|\pi_{ab}(y)| = |y|_a + |y|_b \leqslant |x|_a + |x|_b = |\pi_{ab}(x)|$, so $\pi_{ab}(y)$ is a prefix of $\pi_{ab}(x)$, as desired. ∎

COROLLARY 5.2. *Given words x and y, a multitape Turing machine can test whether x is a congruential prefix of y in time linear in $|xy|$.*

To determine whether a word y is a congruential factor of a word x, it is not sufficient to examine their projections independently, not even their

projections on maximal cliques in (Σ, θ). For example, when $\Sigma = \{a, b, c\}$ and $\theta = \{(a, c), (c, a)\}$, $y = abc$ is not a congruential factor of $x = abbc$, but for each $S \in \{aa, ab, bb, bc, cc\}$, $\pi_S(y)$ is a factor of $\pi_S(x)$. However, by combining information about projections appropriately, whether one word is congruent to a factor of another can be tested in linear time.

THEOREM 5.3. *For a fixed partially commutative alphabet (Σ, θ), given words $x, y \in \Sigma^*$, it can be tested in time linear in $|xy|$ whether y is a congruential factor of x, and, if so, the shortest word z such that $zy \leqslant_\theta x$ will be found.*

Use of the term "the shortest word z" is justified by the proof, in which it is shown that all such shortest words are congruent.

Proof. If, for some $a \in \Sigma$, $|x|_a < |y|_a$, the input is rejected. We may assume that there is no letter that commutes with all other letters: if there is such a letter b, it can be removed from x and y, after checking that $|x|_b \geqslant |y|_b$. Also assume, for convenience, that $\text{alph}(x) = \Sigma$.

For each letter a, let $\Delta(a) = |x|_a - |y|_a$, and let $\Delta = |x| - |y|$.

For distinct letters a and b that do not commute, let

$$P(a, b) = \{(|r|_a, |r|_b) : r\pi_{ab}(y) \text{ is a prefix of } \pi_{ab}(x)\}.$$

Thus, $P(a, b)$ records the *positions* at which the projection $\pi_{ab}(y)$ of y on $\{a, b\}$ occurs in the corresponding projection of x, with each position being given by the letter-count of the string preceding the occurrence of $\pi_{ab}(y)$ in $\pi_{ab}(x)$.

Call a function $f: \Sigma \to \mathbf{N}$ a *location function* (for y in x) if for all $a \in \Sigma$ and all $b \in \bar{\theta}(a)$, the pair $(f(a), f(b))$ is in $P(a, b)$.

CLAIM 1. *There exists a location function for y in x if and only if y is a congruential factor of x.*

Proof. More specifically, location functions give the Parikh mappings of prefixes of x that come before occurrences of y. If $x \equiv uyv$ then $f(a) = |u|_a$, $a \in \Sigma$, specifies a location function: for every pair $(a, b) \in \bar{\theta}$, $\pi_{ab}(uy)$ is a prefix of $\pi_{ab}(x)$ and $|\pi_{ab}(u)|_a = |u|_a$. On the other hand, if f is a location function for y in x, then there exist u and v such that $x \equiv uyv$ and $f(a) = |u|_a$ for all $a \in \Sigma$. To see this, first set, for each $a \in \Sigma$, $r_{aa} = a^{f(a)}$ and $s_{aa} = a^{\Delta(a) - f(a)}$. (Since no letter commutes with all other letters, $\bar{\theta}(a)$ is nonempty for each $a \in \Sigma$, so that $(f(a), f(a')) \in P(a, a')$ for some a' and therefore $f(a) \leqslant \Delta(a)$.) For a pair $(a, b) \in \theta$ such that $a \neq b$, $(f(a), f(b))$ is in $P(a, b)$, so there are words r_{ab} and s_{ab} such that $r_{ab}\pi_{ab}(y)\, s_{ab} = \pi_{ab}(x)$, $|r_{ab}|_a = f(a)$, and $|r_{ab}|_b = f(b)$; note that r_{ab} must be equal to r_{ba}. The families $R = (r_{ab} : (a, b) \in \theta)$ and $S = (s_{ab} : (a, b) \in \theta)$ are

quasi-reconstructible, and $R \cdot \Pi(y) \cdot S = \Pi(x)$, so, applying Proposition 3.2(a) twice, there exist words u and v such that $R = \Pi(u)$, $S = \Pi(v)$, and $uyv \equiv x$. ∎

The question of whether y is congruent to a factor of x can thus be settled by finding a location function. The algorithm here is described in terms of a graph, with edges based on the sets $P(a, b)$; in effect, it moves left-to-right through the projections of x and y, searching for a quasi-reconstructible family of prefixes of the projections of x that precede occurrences of the projections of y. However, only local information at the vertices is needed at each step.

For each $a \in \Sigma$, let

$$V_0(a) = \{k\text{: for some } b \in \bar{\theta}(a) \text{ and some } j, (k, j) \in P(a, b)\};$$

and let

$$V_0 = \{(a, k)\text{: } k \in V_0(a), a \in \Sigma\},$$

$$E_0 = \{\{(a, k), (b, j)\}\text{: } (k, j) \in P(a, b)\},$$

and G_0 be the undirected graph (V_0, E_0).

If f is a location function, the vertices $\{(a, f(a))\text{: } a \in \Sigma\}$ and the edges among them form a subgraph of G_0 that is isomorphic to the graph $(\Sigma, \bar{\theta})$ under the correspondence $a \to (a, f(a))$. The structure of the graph G_0 is restricted by the fact that the edges cannot "cross": if (j_1, k_1) and (j_2, k_2) belong to $P(a, b)$ with $j_1 < j_2$ then $k_1 \leqslant k_2$. (That is, if r_1 and r_2 are prefixes of $\pi_{ab}(x)$ and $|r_1|_a < |r_2|_a$ then $|r_1|_b \leqslant |r_2|_b$.) It is possible that $j_1 = j_2$ and $k_1 < k_2$, but in that case $|y|_a = 0$, since if $r_1 \pi_{ab}(y)\, s_1 = \pi_{ab}(x) = r_2 \pi_{ab}(y)\, s_2$ with $|r_1|_a = |r_2|_a$ and $|r_1|_b < |r_2|_b$ then $r_2 = r_1 b^i$ for some $i \geqslant 1$ and $\pi_{ab}(y)\, s_1 = b^i \pi_{ab}(y)\, s_2$, so that $\pi_{ab}(y)$ can contain only b's.

The following Procedure begins with the graph $G = (V, E)$ equal to G_0. During its operation, vertices (and their incident edges) are deleted from G until either y is "found" in x, or it is determined that y is not congruent to any factor of x. In the description, $V(a)$ denotes the set of integers $\{j\text{: } (a, j) \in V\}$ for the current set of vertices V, and a vertex (a, j) is *saturated* if for all $b \in \bar{\theta}(a)$, (a, j) is currently adjacent to (b, k) for some k.

Procedure

1. Is there some a such that $V(a)$ is empty? If so, halt and reject.

2. Is there some a such that, for $m_a = \min V(a)$, the vertex (a, m_a) is not saturated? If so, delete (a, m_a) and go to (1); otherwise, halt and accept.

CLAIM 2. *If y is a congruential factor of x, then the Procedure halts and*

accepts. When the Procedure halts and accepts, the location function for the first occurrence of y in x is given by $f(a) = \min V(a)$, $a \in \Sigma$.

Proof. If $x \equiv uyv$ for some u and v, then (1) for all $a \in \Sigma$ and all $b \in \bar{\theta}(a)$, $(a, |u|_a)$ and $(b, |u|_b)$ are adjacent in G_0; (2) no vertex $(a, |u|_a)$ is ever deleted; and hence (3) for all $a \in \Sigma$, $(a, |u|_a)$ is always saturated. The Procedure will therefore eventually halt and accept in this case, and at that point, $\min V(a) \leqslant |u|_a$ for all $a \in \Sigma$.

Suppose, on the other hand, that the Procedure halts and accepts, and let $f: \Sigma \to \mathbf{N}$ be the function $f(a) = \min V(a)$ (based on the final sets $V(a)$). Because the Procedure accepted the input, for every a, the vertex $(a, f(a))$ is saturated, so for every $b \in \bar{\theta}(a)$, there is some j such that $(a, f(a))$ is adjacent to (b, j). The vertex $(b, f(b))$ is also saturated, so there is some k such that $(b, f(b))$ is adjacent to (a, k). Since $f(a) = \min V(a)$ and $k \in V(a)$, $f(a) \leqslant k$; similarly, $f(b) \leqslant j$. If $f(a) = k$ then $(a, f(a))$ is adjacent to $(b, f(b))$ and $(f(a), f(b)) \in P(a, b)$; if $f(a) < k$ then $(f(a), j)$ and $(k, f(b))$ are in $P(a, b)$ so $j \leqslant f(b) \leqslant j$, and again $(f(a), f(b)) \in P(a, b)$. This shows that f is a location function, so there exist u_0 and v_0 such that $x \equiv u_0 y v_0$ and, for all a, $f(a) = |u_0|_a$. The word u_0 can be constructed by taking from x the first $f(a)$ occurrences of each letter a (while preserving the relative order of the letters).

To see that $u_0 y$ is the first occurrence of y in x, suppose $x \equiv uyv$ and $|u| \leqslant |u_0|$. As above, when the Procedure halted, no $(a, |u|_a)$ had been deleted and $|u_0|_a = f(a) \leqslant |u|_a$ for all $a \in \Sigma$ (so that the Parikh mapping satisfies $\Psi(u_0) \leqslant \Psi(u)$). Summing over all the letters, $|u_0| \leqslant |u|$, so $|u_0| = |u|$. Moreover, $x \equiv u_0 y v_0 \equiv uyv$ with $\Psi(u_0) \leqslant \Psi(u)$, so (from Proposition 5.1(b)) $u_0 \leqslant_\theta u$; combining this with the length condition, we have $u_0 \equiv u$. Thus, u_0 is a shortest word u such that $uy \leqslant_\theta x$, and all shortest such words are congruent. ∎

CLAIM 3. *The Procedure can be performed in time linear in $|xy|$.*

Proof. We first derive bounds on the size of the graph $G_0 = (V_0, E_0)$. For each $a \in \Sigma$, $|V_0(a)| \leqslant \Delta(a) + 1$, so $|V_0| \leqslant \Delta + |\Sigma|$. (Equality is possible, for example, when $x = y$.) For every pair of letters a and b, $|P(a, b)| \leqslant \Delta(a) + \Delta(b) + 1$, where equality can be achied if $\pi_{ab}(y) = e$ or $\pi_{ab}(y) = \pi_{ab}(x)$. (If, however, every letters occurs in y then $|P(a, b)| \leqslant 1 + \min\{\Delta(a), \Delta(b)\}$ and vertex (a, k) has degree at most $|\bar{\theta}(a)|$.) The size of the edge set E_0 is less than $|\bar{\theta}|/2 + |\Sigma| \cdot \Delta$, and this bound on $|E_0|$ can be derived as follows. Note first that $|\bar{\theta}(a)| \leqslant |\Sigma| - 1$. (When not specified, the sums below are over all letters in Σ.)

$$\sum_a |\bar{\theta}(a)| = |\bar{\theta}| - |\Sigma| \tag{1}$$

$$\sum_a \Delta(a)|\bar{\theta}(a)| \leqslant (|\Sigma| - 1) \cdot \sum_a \Delta(a) = (|\Sigma| - 1) \cdot \Delta \tag{2}$$

$$\sum_{b \in \bar{\theta}(a)} (\Delta(a) + \Delta(b) + 1) = \Delta(a)|\bar{\theta}(a)| + \sum_{b \in \bar{\theta}(a)} \Delta(b) + |\bar{\theta}(a)| \tag{3}$$

Therefore,

$$\begin{aligned} 2\,|E_0| &= \sum_a \sum_{b \in \bar{\theta}(a)} |P(a, b)| \\ &\leqslant \sum_a \sum_{b \in \bar{\theta}(a)} [\Delta(a) + \Delta(b) + 1] \\ &= \sum_a \left[\Delta(a)|\bar{\theta}(a)| + \sum_{b \in \bar{\theta}(a)} \Delta(b) + |\bar{\theta}(a)| \right] \qquad \text{(by (3))}; \end{aligned}$$

and the first two terms expand to the same expression, so

$$\begin{aligned} 2\,|E_0| &\leqslant 2 \sum_c \Delta(c)|\bar{\theta}(c)| + \sum_a |\bar{\theta}(a)| \\ &< 2\,|\Sigma|\,\Delta + |\bar{\theta}| \qquad \text{(by (1) and (2))}. \end{aligned}$$

Since $\Delta = |x| - |y|$ and $|\bar{\theta}| \leqslant |\Sigma|^2$, it follows that $|E_0| < |\Sigma| \cdot |x| + |\Sigma|^2$.

The graph G_0 can be set up, in adjacency-list form, by using a version of the pattern-matching algorithm in the free monoid, one that, given words u and v, finds all words r such that ru is a prefix of v and lists them in increasing order by length; the time required is linear in $|uv|$. The edges in E_0 incident on a vertex (a, k) can be grouped by the letter $b \in \bar{\theta}(a)$, and, within each group, listed in increasing order in the second coordinate. Although the pattern-matching in the free monoids must be done for each pair of distinct letters in $\bar{\theta}$, the total time is linear in $|xy|$ when $|\Sigma|$ is fixed. The subsequent individual costs to find m_a, test if (a, m_a) is saturated, and, if necessary, delete (a, m_a) are constant relative to the length of the input, so the time taken in the Procedure is linear in the number of edges deleted. Since, for a fixed alphabet, $|E_0|$ is bounded above by a linear function of $|x|$, the total time taken is linear in the length of the input. ∎

In the uniform case, in which the input includes the partially commutative alphabet (Σ, θ) as well as the strings x and y, the time taken is at most a constant multiple of $|\bar{\theta}| + |\Sigma| \cdot |x|$ (and it can be assumed that every letter of Σ occurs in x). ∎

6. Conjugacy and Cyclic Equality

This section presents algorithms for testing, in free partially commutative monoids, two generalizations of conjugacy in free monoids. In a free

monoid, words x and y are "conjugate" if $xz = zy$ for some word z, or, equivalently, if $x = uv$ and $y = vu$ for some words u and v (see Lallement, 1979, Cor. 11.5.2; or Lothaire, 1983, Prop. 1.3.4). Conjugacy is an equivalence relation on words, and corresponds to the usual group-theoretic conjugacy relation. The first of these notions of conjugacy can be generalized in a free partially commutative monoid to give rise to an equivalence relation, which is called here "conjugacy." The generalization of the second notion is called here "cyclic equality"; the two relations need not be the same.

Whether words over a partially commutative alphabet (Σ, θ) are conjugate or are cyclically equal can be tested in linear time (Theorem 6.3): both questions are first reduced to pattern-matching problems in $M(\Sigma, \theta)$ and the linear-time algorithm of Section 5 is then applied. Duboc (1986c) has described a process for testing words for conjugacy that deals directly with their projections. The Conjugacy Problem for free partially commutative groups can be reduced to that for free partially commutative monoids and can also be solved in linear time; the reduction generalizes that holding between free groups and free monoids (Wrathall, 1989).

DEFINITION. For words x and y:

(i) x is *cyclically equal* to y (mod θ) if there are words s and t such that $x \equiv st$ and $y \equiv ts$.

(ii) x is *conjugate* to y (mod θ) if there is a word z such that $xz \equiv zy$; such a word z is a *conjugator* of x and y.

The relation of cyclic equality has also been called transposition.

For monoids in general, conjugacy is a reflexive and transitive relation but need not be symmetric, and cyclic equality is reflexive and symmetric but need not be transitive; also, cyclically equal elements are conjugate. Otto (1984) has considered these and other possible definitions of conjugacy in arbitrary monoids. In abelian monoids as well as free monoids, the two relations coincide, but they do not coincide in general for free partially commutative monoids. Consider, for example, $\Sigma = \{a, b, c\}$ and $\theta = \{(a, c), (c, a)\}$, so that b commutes with neither a nor c, and let $x = abc$ and $y = cba$. Then $x \cdot aba = ab\underline{ca}ba \equiv ab\underline{ac}ba = aba \cdot y$, showing that x and y are conjugate, but, since the congruence class of x is $\{x\}$, the only words cyclically equal to x are abc, cab, acb, bca, and bac. Note also that, although x and y are not cyclically equal, their projections on each two-letter alphabet are cyclically equal in the free monoids.

In the case of free partially commutative monoids, Duboc has established the following connection between the two relations. It follows from this characterization that conjugacy mod θ is a symmetric relation and is the transitive closure of cyclic equality mod θ.

Notation. Let CE denote the binary relation of cyclic equality mod θ, and CE^k the composition of CE with itself k times. That is, CE^0 is the identity relation and, for $k \geqslant 0$, $CE^{k+1} = \{(x, y)$: there is some w such that x is cyclically equal to w (mod θ) and $(w, y) \in CE^k\}$.

PROPOSITION 6.1. (Duboc, 1986b). *For any words x and y, x is conjugate to y (mod θ) if and only if there is some $k < |\Sigma|$ such that $(x, y) \in CE^k$.*

The bound on the number of compositions of CE is, more precisely, the maximum diameter of the connected components of the undirected graph $(\Sigma, \tilde{\theta})$; that maximum must be less than the number of vertices in the graph and the bound $|\Sigma|$ is sufficient for the purpose here.

The following proposition reduces questions of cyclic equality and conjugacy to the checking of letter-counts and questions of pattern-matching.

PROPOSITION 6.2. *For any words x and y:*

(a) *x is cyclically equal to y (mod θ) if and only if $\Psi(x) = \Psi(y)$ and y is congruent to a factor of x^2;*

(b) *x is conjugate to y (mod θ) if and only if $\Psi(x) = \Psi(y)$ and for some $n \leqslant |\Sigma|$, y is congruent to a factor of x^n.*

In a free monoid, the condition that the Parikh mappings are the same can be weakened to requiring that the two words have the same length. The bound of $|\Sigma|$ in the second part is achievable: continuing the example above (with $x = abc$ and $y = cba$), x^3 is congruent to $aba \cdot y \cdot cbc$, but no word congruent to x or x^2 has y as a factor.

Proof. The proof of each part uses the following facts.

CLAIM 1. *If $(x, y) \in CE^k$, $k \geqslant 0$, then y is congruent to a factor of x^{k+1}.*

Proof. The statement is clearly true for $k = 0$. Continuing by induction, suppose $k \geqslant 1$ and $(x, y) \in CE^k$, so that there are w, s, and t such that $x \equiv st$, $w \equiv ts$, and $(w, y) \in CE^{k-1}$. Then $w^k \equiv uyv$ for some u and v, so $x^{k+1} \equiv (st)^{k+1} = s(ts)^k t \equiv sw^k t \equiv (su)\, y(tv)$, and hence y is congruent to a factor of x^{k+1}. ∎

CLAIM 2. *If $\Psi(x) = \Psi(y)$ and $x^n \equiv uyv$, $n \geqslant 1$, then $x^{n-1} \equiv uv$ and $xu \equiv uy$.*

Proof. If $n = 1$ then (since $\Psi(x) = \Psi(y)$) $x \equiv y$ and $u = v = e$, so the conclusion holds. If $n > 1$ then $x^n = x^{n-1} \cdot x \equiv (u)(yv)$ with $\Psi(u) \leqslant \Psi(uv) = \Psi(x^{n-1})$, so, from Proposition 5.1(b), there is some r such that $x^{n-1} \equiv ur$ and $rx \equiv yv$. From $x^n = x \cdot x^{n-1} \equiv (uy)(v)$, it follows similarly that

$x^{n-1} \equiv sv$ for some s, with $xs \equiv uy$. Combining these, $x^{n-1} \equiv ur \equiv sv$, where $\Psi(u) = \Psi(uy) - \Psi(y) = \Psi(xs) - \Psi(x) = \Psi(s)$, so $u \equiv s$ and $r \equiv v$; hence $x^{n-1} \equiv ur \equiv uv$ and $xu \equiv xs \equiv uy$. ∎

If $\Psi(x) = \Psi(y)$ and y is congruent to a factor of x^n then (from Claim 2) there are u and v such that $x^{n-1} \equiv uv$ and $xu \equiv uy$, so x and y are conjugate; when $n = 2$, $x \equiv uv$ and $uvu \equiv uy$, so in addition $y \equiv vu$ and x is cyclically equal to y. Conversely, if x and y are cyclically equal or conjugate then clearly $\Psi(x) = \Psi(y)$. If x is cyclically equal to y then (from Claim 1, with $k = 1$) y is congruent to a factor of x^2. If x is conjugate to y, then, from the previous proposition, $(x, y) \in \mathrm{CE}^k$ for some $k < |\Sigma|$, so (again from Claim 1) y is congruent to a factor of x^{k+1} with $k + 1 \leqslant |\Sigma|$. ∎

THEOREM 6.3. *For a fixed partially commutative alphabet (Σ, θ) there are linear-time algorithms for the following problems.*

(1) *Given words x and y, test if they are cyclically equal and, if so, produce a pair of words (s, t) such that $x \equiv st$ and $y \equiv ts$.*

(2) *Given words x and y, test if they are conjugate, and, if so, produce a shortest conjugator.*

Proof. From Proposition 6.2, each of these problems reduces to testing, first, whether the two words have the same Parikh mapping, and, second, whether one of the words is congruent to a factor of a certain power of the other. Testing whether $\Psi(x) = \Psi(y)$ can clearly be done in time linear in $|xy|$ (even on a Turing machine); from Theorem 5.3, testing whether y is congruent to a factor of x^q and, if so, finding the "first" location of y in x^q can be done in time linear in $|yx^q| = |y| + q|x|$ and so is linear in $|xy|$ when q is fixed. In case (1), that is, cyclic equality, if y is congruent to a factor of x^2 then a shortest word u such that $uy \leqslant_\theta x^2$ will be returned. A word v such that $x \equiv uv$ can be easily found and then, as in Claim 2 of Proposition 6.2, (u, v) is a pair such that $x \equiv uv$ and $y \equiv vu$.

In the case of conjugacy, if y is congruent to a factor of x^L for $L = |\Sigma|$, then a shortest word u such that $uy \leqslant_\theta x^L$ will be returned; in fact, that word u must be a shortest conjugator of x and y. As in the proof of Proposition 6.2, $xu \equiv uy$, so that u is a conjugator; also, $x^{L-1} \equiv uv$, where v is such that $x^L \equiv uyv$.

Suppose u' is any conjugator of x and y: $xu' \equiv u'y$. Note first that there is some n such that $u' \leqslant_\theta x^n$: for each pair $(a, b) \in \bar{\theta}$, $\pi_{ab}(u')$ is a conjugator of $\pi_{ab}(x)$ and $\pi_{ab}(y)$ in the free monoid $\{a, b\}^*$, so there exist an integer $k(ab) \geqslant 0$ and a prefix w_{ab} of $\pi_{ab}(x)$ such that $\pi_{ab}(u') = (\pi_{ab}(x))^{k(ab)} w_{ab}$ (Lallement, 1979, Lemma 11.5.1; Lothaire, 1983, Prop. 1.3.4). Taking n to be large than any of the integers $\{k(ab): (a, b) \in \bar{\theta}\}$, each projection of u' is

a prefix of the corresponding projection of x^n, and the assertion follows from Proposition 5.1(a). (If no proper suffix of u' is also a conjugator, then n need be no larger than $|\Sigma| - 1$ (Duboc, 1986b).)

Let v' be a word such that $x^n \equiv u'v'$. If $n \leqslant L-1$ then $u'y$ $(\equiv xu')$ is a congruential prefix of x^L, so since u is a shortest word such that $uy \leqslant_\theta x^L$, $|u'| \geqslant |u|$. If $n \geqslant L$ then $u'v' \equiv x^n \equiv u(vx^{n-L+1})$, so there exist words r, s, and t such that $u' = rs$, $u \equiv rt$, and $\text{alph}(s) \times \text{alph}(t) \subseteq \theta$ (Cori and Perrin, 1985, Prop. 1.3). From $xu \equiv uy$ an $xu' \equiv u'y$, we can conclude that $rtys \equiv xrts \equiv xrst \equiv rsyt$, so $tys \equiv syt$. Since $\text{alph}(s) \times \text{alph}(t) \subseteq \theta$, it follows from Proposition 3.1 that $ty \equiv yt$: for any $(a, b) \in \bar{\theta}$, if $|t|_a = |t|_b = 0$ then $\pi_{ab}(ty) = \pi_{ab}(y) = \pi_{ab}(yt)$; and if either $|t|_a$ or $|t|_b$ is positive then neither a nor b can be in $\text{alph}(s)$, so $\pi_{ab}(s) = e$ and $\pi_{ab}(ty) = \pi_{ab}(tys) = \pi_{ab}(syt) = \pi_{ab}(yt)$. Therefore, $x^L \equiv uyv \equiv rtyv \equiv rytv$, so, since u is a shortest prefix of x^L with this property, $r = u$, $t = e$, $u' \equiv us$, and again $|u'| \geqslant |u|$. ∎

RECEIVED February 4, 1988; FINAL MANUSCRIPT RECEIVED July 26, 1989

REFERENCES

AALBERSBERG, IJ., AND ROZENBERG, G. (1988), Theory of traces, *Theoret. Comput. Sci.* **60**, 1–82.

AHO, A. V., HOPCROFT, J. E., AND ULLMAN, J. D. (1974), "The Design and Analysis of Computer Algorithms," Addison–Wesley, Reading, MA.

AVENHAUS, J., AND MADLENER, K. (1980), String matching and algorithmic problems in free groups, *Rev. Colombiana Mat.* **14**, 1–16.

BERTONI, A., MAURI, G., AND SABADINI, N. (1982), Equivalence and membership problems for regular trace languages, *in* "Lecture Notes in Computer Science," Vol. 140, pp. 61–71, Springer-Verlag, Berlin/New York.

BOOK, R. V., AND LIU, H.-N. (1987), Word problems and rewriting in a free partially commutative monoid, *Inform. Process. Lett.* **26**, 29–32.

CARTIER, P., AND FOATA, D. (1969), Problemes combinatoires de commutation et rearrangements, *in* "Lecture Notes in Mathematics," Vol. 85, Springer-Verlag, Berlin/ New York.

CORI, R., AND METIVIER, Y. (1985), Recognizable subsets of some partially abelian monoids, *Theoret. Comput. Sci.* **35**, 179–189.

CORI, R., AND PERRIN, D. (1985), Automates et commutations partielles, *RAIRO Inform. Théor.* **19**, 21–32.

DUBOC, C. (1986a), Some properties of commutations in free partially commutative monoids, *Inform. Process. Lett.* **20**, 1–4.

DUBOC, C. (1986b), On some equations in free partially commutative monoids, *Theoret. Comput. Sci.* **46**, 159–174.

DUBOC, C. (1986c), "Commutations dans les monoides libres," Thesis, Univ. Rouen.

FLÉ, M., AND ROUCAIROL, G. (1985), Maximal serializability of iterated transactions, *Theoret. Comput. Sci.* **38**, 1–16.

GALIL, Z., AND SEIFERAS, J. (1983), Time–space optimal string matching, *J. Comput. System Sci.* **26**, 280–294.

KELLER, R. (1973), Parallel program schemata and maximal parallelism, *J. Assoc. Comput. Mach.* **20**, 514–537.

KNUTH, D. E., MORRIS, J. H., AND PRATT, V. R. (1977), Fast pattern-matching in strings, *SIAM J. Comput.* **6**, 323–350.

LALLEMENT, G. (1979), "Semigroups and Combinatorial Applications," Wiley, New York.

LOTHAIRE, M. (1983), "Combinatorics on Words," Addison–Wesley, Reading, MA.

OTTO, F. (1984), Conjugacy in monoids with a special Church-Rosser presentation is decidable, *Semigroup Forum* **29**, 223–240.

PERRIN, D. (1966), Words over a partially commutative alphabet, *in* "Combinatorial Algorithms on Words" (A. Apostolico and Z. Galil, Eds.), NATO ASI Ser., Vol. F12, pp. 329–340, Springer-Verlag, Berlin/New York.

WRATHALL, C. (1989), Free partially commutative groups, *in* "Combinatorics, Computing and Complexity" (D.-Z. Du and G. Hu, Eds.), pp. 195–216, Kluwer Academic/Science Press, Norwell, MA.